

AWS Serverless for Data Warehousing

Minh Dinh

1. Introduction

Setting up an on-premise data warehouse seems to be costly at the beginning. Various cost components must be planned and preserved beforehand, including infrastructure investment, human resources, and operational costs. For example, a data warehouse requires to be built on a physical server, which can cost from 5,000 to 15,000 USD, (Stanovici, 2022b), adding with operation system of Windows Server Standard edition, which is likely worth 1069\$, (Windows Server 2022 Licensing & Pricing | Microsoft, n.d.), and license of a Microsoft SQL Server Enterprise database, costing around 15,123 USD, (SQL Server 2022—Pricing | Microsoft, n.d.). In total, the up-front cost can be more than 20,000 USD. Other than that, companies also need to hire high-paid IT personnel to take care of those servers and prepare a budget for operational costs such as power usage, and maintenance on a monthly basis. And, these costs can become waste when your team changes the technology or your data warehouse actually require less computing power than the server provides. To avoid these server management costs, corporates should consider leveraging cloud serverless services, which generate costs on what your team actually uses. With cloud serverless solutions, the physical servers are still there, however, they are all managed by cloud providers, while the developers only concentrate on their main tasks of generating application codes. To illustrate this benefit of cloud technology, I would like to demonstrate building a data pipeline using AWS serverless services.

2. Tools Description

AWS (Amazon Web Services) is a cloud computing platform provided by the giant e-commerce company Amazon. AWS provides various services to allow users to develop application and software infrastructure such as computing, storage, and databases, (What-is-aws, n.d.), which can be leveraged with other provided technologies such as artificial intelligence, and the Internet of Things to be able to build big data applications at ease.

To illustrate the application of AWS serverless technologies in developing data pipelines to build a data warehouse. The later experiment will leverage the following services:

- **IAM:** The service manages users, roles, and permissions, ensuring only authorized access can communicate with AWS infrastructure, (What Is IAM? - AWS Identity and Access Management, n.d.).
- **ECR:** It is a fully managed Docker container registry service provided by AWS. It allows developers to store Docker container images on AWS cloud infrastructure, (What Is Amazon Elastic Container Registry? - Amazon ECR, n.d.).
- **ECS:** The service allows users to run Docker containers across a cluster. The infrastructure of a cluster can be either EC2 instances (physical servers) or AWS Fargate (a serverless solution), (What Is Amazon Elastic Container Service? - Amazon Elastic Container Service, n.d.).
- **Lambda:** A serverless computing service that allows users to run code and develop applications (supporting Python, Java, and Javascript) without provisioning or managing servers, (What Is AWS Lambda? - AWS Lambda, n.d.-b).
- **Cloudwatch:** The service allows users to monitor and log the activities of AWS resources, and services, (What Is Amazon CloudWatch? - Amazon CloudWatch, n.d.).
- **EventBridge:** The service allows users to integrate different AWS services to build a whole event-driven application, (What Is Amazon EventBridge? - Amazon EventBridge, n.d.).
- **SNS:** The service allows users to send messages or notifications to recipients via email, (What Is Amazon SNS? - Amazon Simple Notification Service, n.d.).
- **S3:** An object storage service, which allows users to store different types of objects (e.g. images, videos, CSV, parquet, ...) and offers unlimited storage. The service can be used for backup, data archiving, and big data analytics, (What Is Amazon S3? - Amazon Simple Storage Service, n.d.).
- **Glue:** The service allows users to migrate data between different sources. The service is used to clean, and transform data for analysis, (What Is AWS Glue? - AWS Glue, n.d.).
- **Aurora:** The service is a fully managed relational database, which is compatible with MySQL and PostgreSQL. The service can be used to implement a data warehouse or online analytical processing (OLAP) system, (What Is Amazon Aurora? - Amazon Aurora, n.d.).

3. Objective

The later experiment will evaluate two things: (1) the estimated incurred cost, and (2) the ease of implementation of the AWS serverless solution in terms of developing and

maintaining a simple data warehouse, which stores the transformed data scraped from a website on the internet with the data size under 1 MB. These two aspects are also discussed and compared with the on-premise solution to highlight the features of AWS serverless technologies.

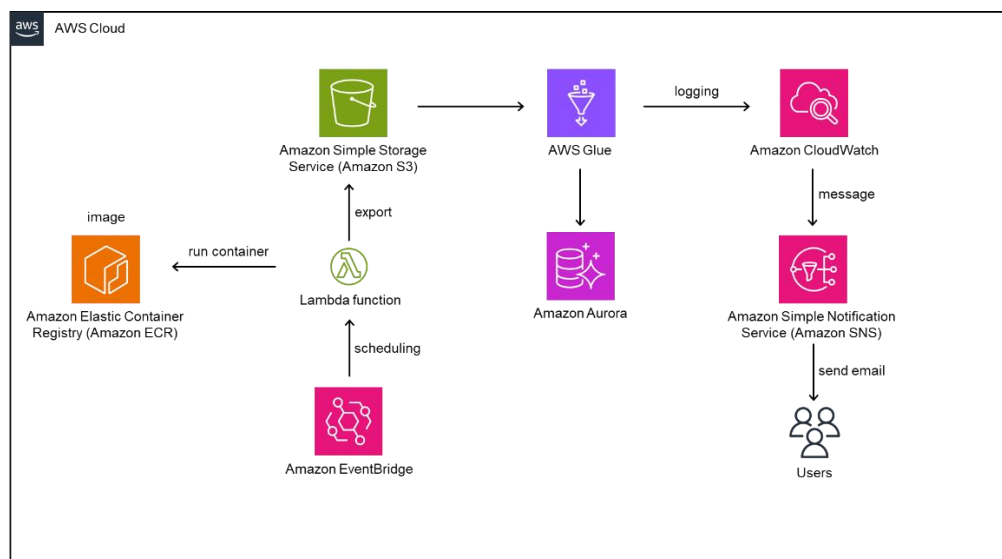
4. Design of Experiments

An AWS serverless solution is introduced to load data scraped from a website to the data warehouse by schedules. The data is scraped from the following URL address: <https://s.cafef.vn/lich-su-giao-dich-vnindex-1.chn#data>, the website shows a table recording data of Vietnam's stock index updated daily. Each row has the following values: date, closing index, adjusted index, index's percent changes, matched volume, matched amount, negotiated volume, negotiated amount, opening index, highest index, and lowest index.

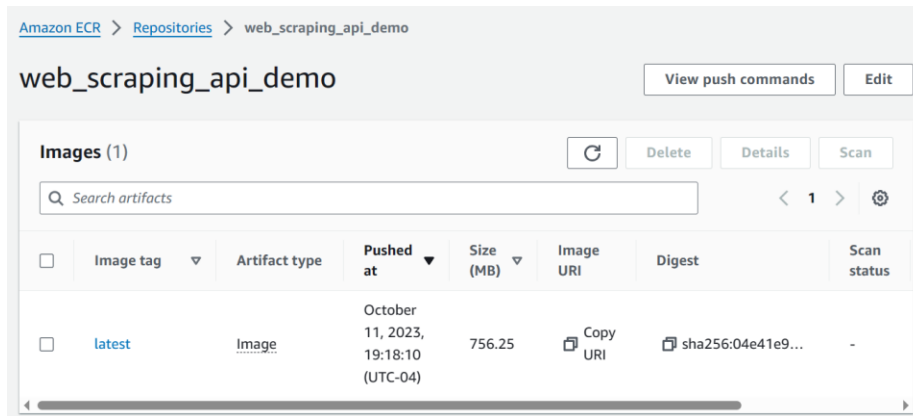
| Ngày | Giá (nghìn VND) | | Thay đổi | GD khớp lệnh | | GD thỏa thuận | | Giá (nghìn VND) | | |
|------------|-----------------|------------|-----------------|--------------|------------------|---------------|------------------|-----------------|----------|-----------|
| | Đóng cửa | Điều chỉnh | | Khối lượng | Giá trị (tỷ VND) | Khối lượng | Giá trị (tỷ VND) | Mở cửa | Cao nhất | Thấp nhất |
| 25/10/2023 | 1101.66 | 1,101.66 | -4.24(-0.38 %) | 483,430,997 | 9,692.92 | 54,812,103 | 1,302.47 | 1,106.41 | 1,111.39 | 1,100.64 |
| 24/10/2023 | 1105.9 | 1,105.9 | 12.37(1.13 %) | 444,069,106 | 9,109.94 | 60,481,790 | 1,299.38 | 1,095.42 | 1,107.3 | 1,088.17 |
| 23/10/2023 | 1093.53 | 1,093.53 | -14.5(-1.31 %) | 500,481,939 | 10,110.02 | 62,036,531 | 1,966.32 | 1,106.56 | 1,106.8 | 1,086.54 |
| 20/10/2023 | 1108.03 | 1,108.03 | 20.18(1.86 %) | 592,521,786 | 12,391.73 | 65,671,082 | 1,779.05 | 1,085.83 | 1,108.16 | 1,073.73 |
| 19/10/2023 | 1087.85 | 1,087.85 | -15.55(-1.41 %) | 615,816,621 | 12,695.44 | 31,159,234 | 934.32 | 1,102.93 | 1,103.65 | 1,087.85 |

With this experiment, the new record in a day will be daily imported and stored in AWS Aurora (a serverless database). The whole process will be done by using AWS serverless services without the involvement of managing physical servers.

The experiment will be implemented with the following architecture and steps:

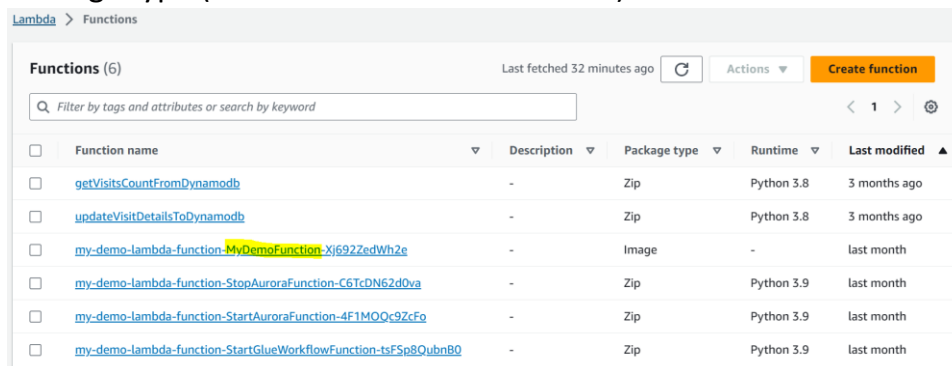


- **Step 1:** A Docker image of containerizing an API application, which allows users to send a GET request to retrieve a new daily record, is built. The image is then pushed to ECR.



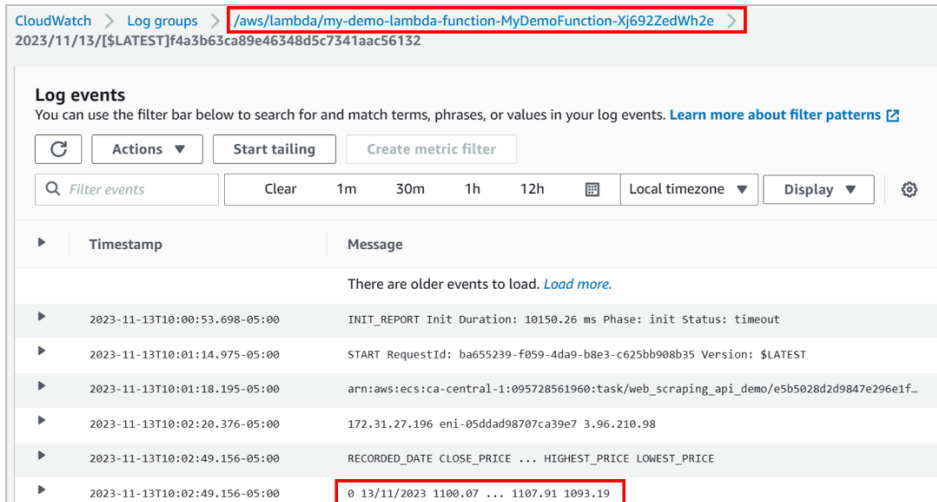
- **Step 2:** Develop a Lambda function to run the container from the uploaded image to launch the API. Then, the Lambda function performs a GET request to retrieve the scraped data and save the data into S3.

In this experiment, the function MyDemoFunction was developed in Image Package type (as the size exceeded 250 MB).

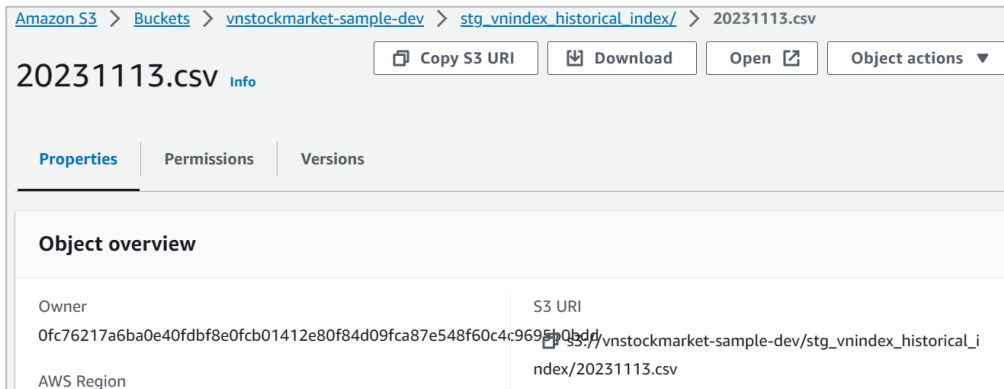


The function's code can be viewed in the Appendix, 8.1.

For example, on 13th November 2023, the function was invoked to retrieve the data for the day.



Then, the output file for the day can be found in S3 bucket vnstockmarket-sample-dev with the name “stg_vnindex_historical_index/20231113.csv” as follows:



The data was correctly recorded in the csv file “20231113.csv”:

https://s.cafef.vn/lich-su-giao-dich-vnindex-1.chn#data

Lịch sử giá Thông kê đặt lệnh Khối ngoại Tự doanh Khớp lệnh theo phiên Cổ đông & Nội bộ

Mã chứng khoán: VNINDEX Thời gian: 13/11/2023 - 13/11/2023

Xem Xuất Excel

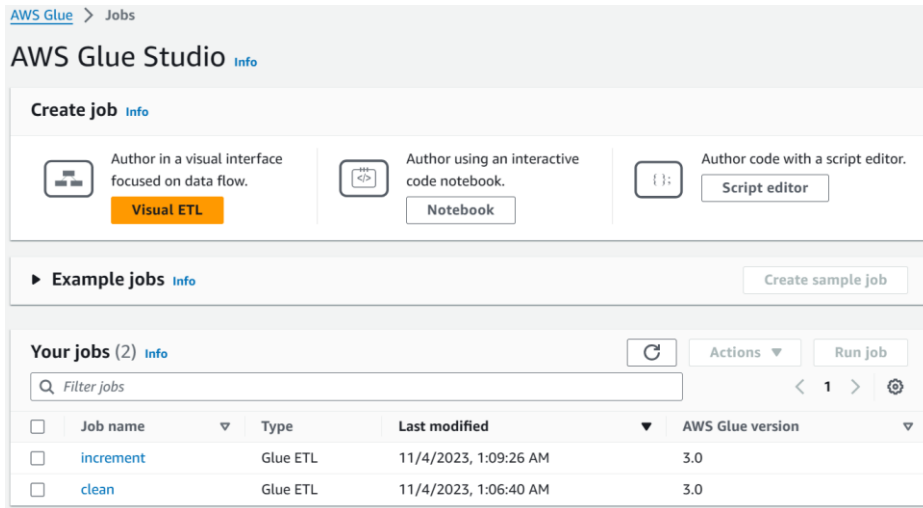
| Ngày | Giá (nghìn VND) | | Thay đổi | GD khớp lệnh | | GD thỏa thuận | | Giá (nghìn VND) | | |
|------------|-----------------|------------|----------------|--------------|------------------|---------------|------------------|-----------------|----------|-----------|
| | Đóng cửa | Điều chỉnh | | Khối lượng | Giá trị (tỷ VND) | Khối lượng | Giá trị (tỷ VND) | Mở cửa | Cao nhất | Thấp nhất |
| 13/11/2023 | 1100.07 | 1,100.07 | -1.61(-0.15 %) | 740,255,909 | 14,653.41 | 48,147,673 | 1,359.48 | 1,103.66 | 1,107.91 | 1,093.19 |

20231113.csv - Excel

| | A | B | C | D | E | F | G | H | I | J |
|---|---------------|-------------|----------------|---------------|---------------|--------------------|--------------------|------------|---------------|--------------|
| | RECORDED_DATE | CLOSE_PRICE | INDEX_CHANGE | MACHED_VOLUME | MACHED_AMOUNT | SELF_DEALED_VOLUME | SELF_DEALED_AMOUNT | OPEN_PRICE | HIGHEST_PRICE | LOWEST_PRICE |
| 1 | 13/11/2023 | 1100.07 | -1.61(-0.15 %) | 740255909 | 14653.41 | 48147673 | 1359.48 | 1103.66 | 1107.91 | 1093.19 |

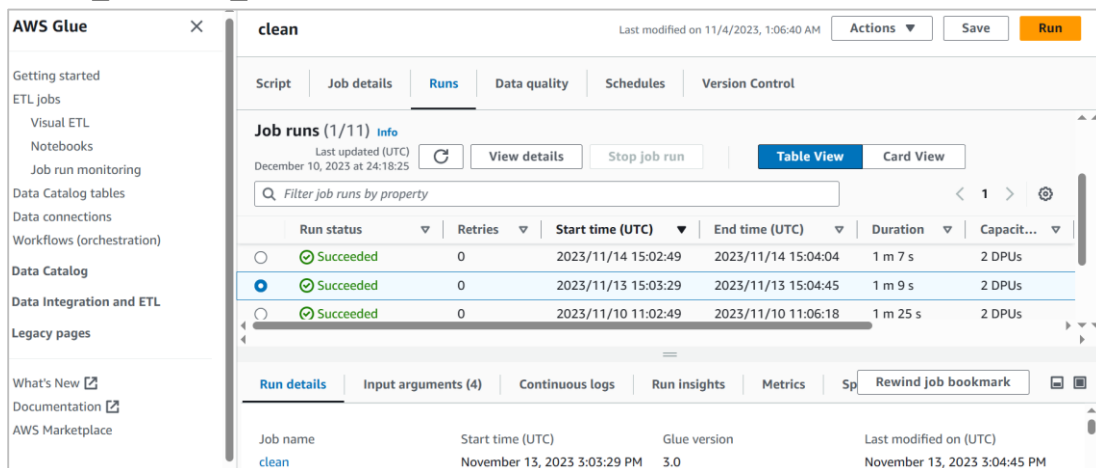
- **Step 3:** Create Glue PySpark jobs to load the data from S3, clean the data, and import it into Aurora.

In this experiment, two PySpark jobs named “clean”, and “increment” are created.



And the codes of the two jobs can be viewed in the Appendix, 8.2 and 8.3.

For example, on 13th November 2023, the “clean” job was run successfully after being fired by S3 Event Notifications as a new data file created in S3. The job split the column INDEX_CHANGE into two other columns INDEX_CHANGE_VALUE and INDEX_CHANGE_PERCENT.



And, the output data was exported to csv file “cleaned/stg_vnindex_historical_index/20231113.csv” in vnstockmarket-sample-dev bucket.

Amazon S3 > Buckets > vnstockmarket-sample-dev > cleaned/ > stg_vnindex_historical_index/ > 20231113.csv

20231113.csv [Info](#)

[Copy S3 URI](#) [Download](#) [Open](#) [Object actions](#)

[Properties](#) [Permissions](#) [Versions](#)

Object overview

| | |
|--|---|
| Owner | S3 URI |
| Ofc76217a6ba0e40fdbf8e0fcb01412e80f84d09fca87e548f60c4c9695b0bdf | s3://vnstockmarket-sample-dev/cleaned/stg_vnindex_historical_index/20231113.csv |
| AWS Region | Amazon Resource Name (ARN) |
| Canada (Central) ca-central-1 | |

20231113.csv - Excel

| RECORDED_DATE | INDEX_SCORE | INDEX_CHANGE_VALUE | INDEX_CHANGE_PERCENT | MACHED_VOLUME | MACHED_AMOUNT | SELF_DEALED_VOLUME | SELF_DEALED_AMOUNT | OPEN_PRICE | HIGHEST_PRICE |
|---------------|-------------|--------------------|----------------------|---------------|---------------|--------------------|--------------------|------------|---------------|
| 11/13/2023 | 1100.07 | -1.61 | -0.15 | 740255909 | 14653.41 | 48147673 | 1359.48 | 1103.66 | 1103.66 |

After the “clean” job was completed, the “increment” job was triggered to import the cleaned data into Aurora Serverless.

AWS Glue [×](#) **increment** Last modified on 11/4/2023, 1:09:26 AM [Actions](#) [Save](#) [Run](#)

[Script](#) [Job details](#) [Runs](#) [Data quality](#) [Schedules](#) [Version Control](#)

Job runs (1/12) [Info](#)

Last updated (UTC) December 10, 2023 at 01:04:12 [View details](#) [Stop job run](#) [Table View](#) [Card View](#)

| Run status | Retries | Start time (UTC) | End time (UTC) | Duration | Capacit... |
|------------|---------|---------------------|---------------------|----------|------------|
| Succeeded | 0 | 2023/11/14 15:04:35 | 2023/11/14 15:06:18 | 1 m 26 s | 2 DPU |
| Succeeded | 0 | 2023/11/13 15:05:16 | 2023/11/13 15:07:10 | 1 m 17 s | 2 DPU |

- **Step 4:** Create an SNS topic to send the Cloudwatch logs of the data pipeline (success/failure) to users’ email

In this experiment, a SNS topic named glue-jobs-failed-notification was created, which was able to send a notification about the error of a failed Glue job to an email address: test1dbmail@gmail.com.

Amazon SNS > Topics > glue-jobs-failed-notification

glue-jobs-failed-notification

Edit Delete Publish message

Details

| | |
|--|---|
| Name glue-jobs-failed-notification | Display name glue-jobs-failed-notification |
| ARN arn:aws:sns:ca-central-1:095728561960:glue-jobs-failed-notification | Topic owner 095728561960 |
| Type Standard | |

[Subscriptions](#) | [Access policy](#) | [Data protection policy](#) | [Delivery policy \(HTTP/S\)](#) | [Delivery status logging](#) | [Encryption](#) | [Tags](#) | [Integrations](#)

Subscriptions (1) Edit Delete Request confirmation Confirm subscription Create subscription

| ID | Endpoint | Status | Protocol |
|-------------------------------------|-----------------------|-----------|----------|
| fa238d3a-e743-4415-9f11-42ec0143... | test1dbmail@gmail.com | Confirmed | EMAIL |

For example, on 8th November 2023, there was a failure of “increment” Glue job, saying that the file 20231108.csv did not exist, and this error was sent directly to the email: test1dbmail@gmail.com as well. The difference of reported time was caused by different time zones between mail server and AWS region.

increment

Last modified on 11/4/2023, 1:09:26 AM Actions Save Run

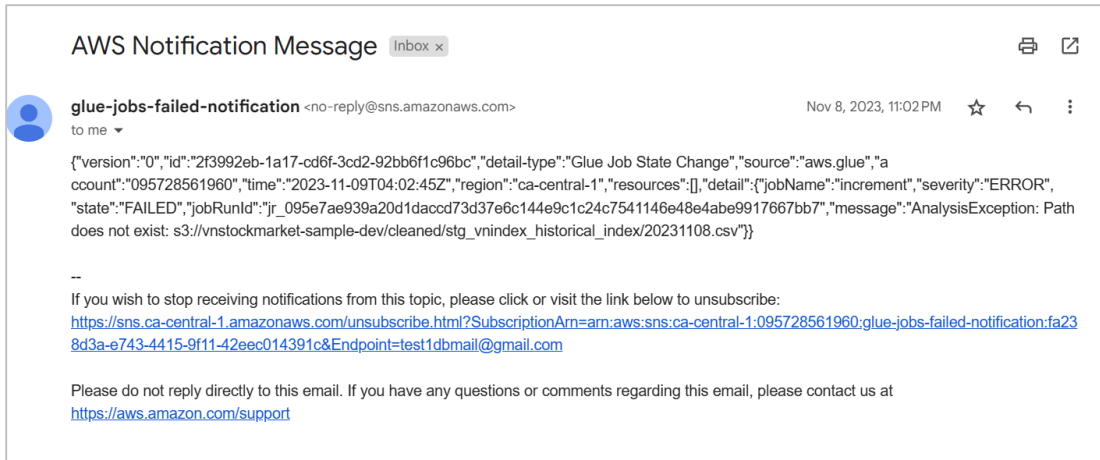
Script Job details **Runs** Data quality Schedules Version Control

Job runs (1/12) Info
Last updated (UTC)
December 10, 2023 at 01:31:16 View details Stop job run Table View Card View

| Run status | Retries | Start time (UTC) | End time (UTC) | Duration | Capacit... |
|-------------|---------|---------------------|---------------------|----------|------------|
| ✓ Succeeded | 0 | 2023/11/09 11:04:33 | 2023/11/09 11:06:02 | 1 m 13 s | 2 DPU |
| ✗ Failed | 0 | 2023/11/09 04:00:38 | 2023/11/09 04:01:45 | 40 s | 2 DPU |

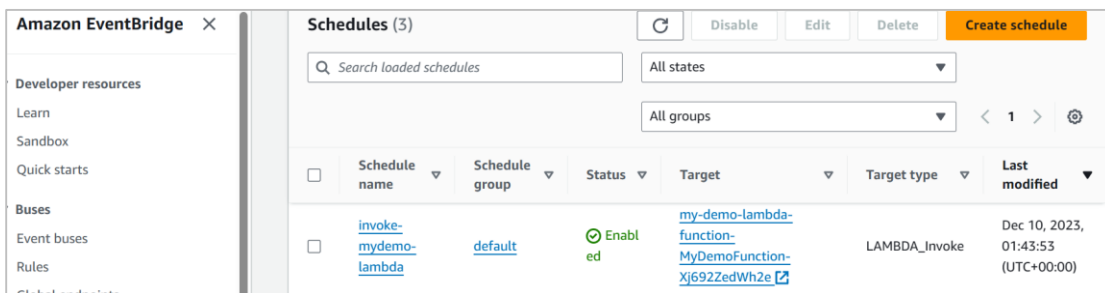
Run details | Input arguments (7) | Continuous logs | Run insights | Metrics | Sp | Rewind job bookmark

✗ AnalysisException: Path does not exist: s3://vnstockmarket-sample-dev/cleaned/stg_vnindex_historical_index/20231108.csv



- **Step 5: Scheduling the whole pipeline using EventBridge.**

In this experiment, a schedule named invoke-mydemo-lambda was created.



The schedule was configured to run at 10 pm (UTC+7) on a daily basis and trigger the whole data pipeline processing the mentioned steps.

Cron expression [Info](#)

| | | | | | |
|---------|-------|--------------|-------|-------------|------|
| 0 | 22 | ? | * | 2-6 | * |
| Minutes | Hours | Day of month | Month | Day of week | Year |

Copy cron expression

Next 10 trigger dates

Date and time are displayed in the selected time zone for which this schedule is set in UTC format, e.g. "Wed, Nov 9, 2022 09:00 (UTC - 08:00)"

Mon, 11 Dec 2023 22:00:00 (UTC+07:00)
Tue, 12 Dec 2023 22:00:00 (UTC+07:00)
Wed, 13 Dec 2023 22:00:00 (UTC+07:00)
Thu, 14 Dec 2023 22:00:00 (UTC+07:00)
Fri, 15 Dec 2023 22:00:00 (UTC+07:00)
Mon, 18 Dec 2023 22:00:00 (UTC+07:00)
Tue, 19 Dec 2023 22:00:00 (UTC+07:00)
Wed, 20 Dec 2023 22:00:00 (UTC+07:00)
Thu, 21 Dec 2023 22:00:00 (UTC+07:00)
Fri, 22 Dec 2023 22:00:00 (UTC+07:00)

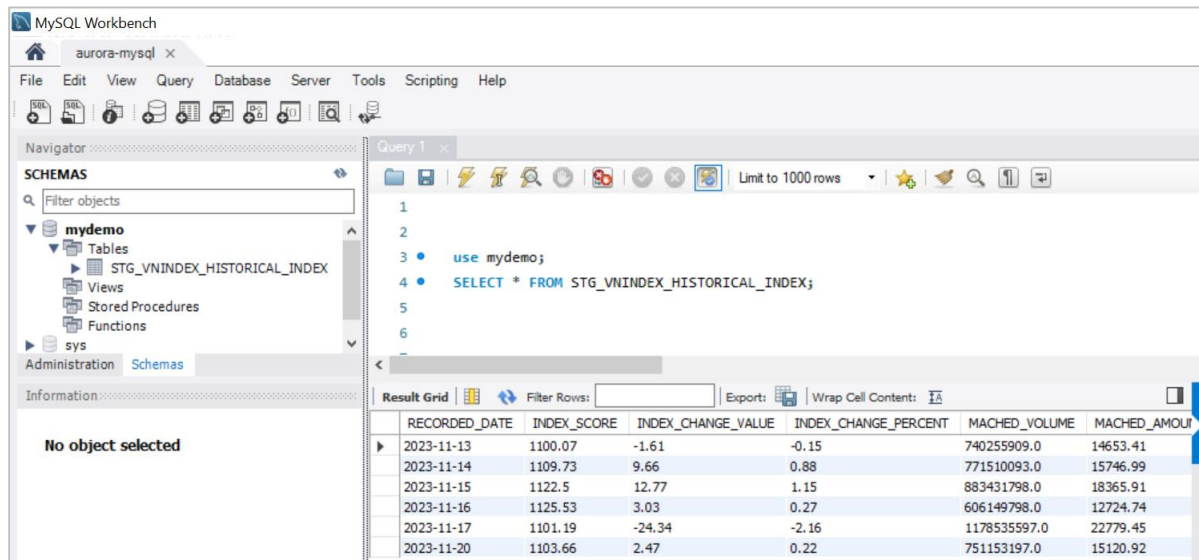
5. Results

- First, our data warehousing solution is successfully implemented with AWS serverless services.

| Log streams (6) | | | Delete | Create log stream | Search all log streams |
|--|---|--------------------------------------|---------------------------------------|----------------------|------------------------|
| <input type="text" value="Filter log streams or try prefix search"/> | | <input type="checkbox"/> Exact match | <input type="checkbox"/> Show expired | Info | < 1 > |
| <input type="checkbox"/> | Log stream | Last event time | | | |
| <input type="checkbox"/> | 2023/11/20/[\$LATEST]b2df00b654b449edbe6cca99d5... | 2023-11-20 10:02:10 (UTC-05:00) | | | |
| <input type="checkbox"/> | 2023/11/17/[\$LATEST]90a7760f19974cfd93036fd482c... | 2023-11-17 10:02:17 (UTC-05:00) | | | |
| <input type="checkbox"/> | 2023/11/16/[\$LATEST]c2fbe792d5114e13a3082f0daa6... | 2023-11-16 10:02:08 (UTC-05:00) | | | |
| <input type="checkbox"/> | 2023/11/15/[\$LATEST]ef6c29783c43408ca322f128fe8... | 2023-11-15 10:02:10 (UTC-05:00) | | | |
| <input type="checkbox"/> | 2023/11/14/[\$LATEST]c0501fd8b7554a07bad9f77108a... | 2023-11-14 10:02:10 (UTC-05:00) | | | |
| <input type="checkbox"/> | 2023/11/13/[\$LATEST]f4a3b63ca89e46348d5c7341aac... | 2023-11-13 10:02:50 (UTC-05:00) | | | |

From 13 to 20 November 2023, the pipeline was executed automatically at 10:00 pm (UTC-5).

And, the associated data was successfully transformed and imported into the Aurora Serverless – MySQL.



The screenshot shows the MySQL Workbench interface with a query executed against the 'mydemo' database. The query is 'SELECT * FROM STG_VNINDEX_HISTORICAL_INDEX;'. The result grid displays data for the period from 2023-11-13 to 2023-11-20. The columns in the result grid are: RECORDED_DATE, INDEX_SCORE, INDEX_CHANGE_VALUE, INDEX_CHANGE_PERCENT, MACHED_VOLUME, and MACHED_AMOUI.

| RECORDED_DATE | INDEX_SCORE | INDEX_CHANGE_VALUE | INDEX_CHANGE_PERCENT | MACHED_VOLUME | MACHED_AMOUI |
|---------------|-------------|--------------------|----------------------|---------------|--------------|
| 2023-11-13 | 1100.07 | -1.61 | -0.15 | 740255909.0 | 14653.41 |
| 2023-11-14 | 1109.73 | 9.66 | 0.88 | 771510093.0 | 15746.99 |
| 2023-11-15 | 1122.5 | 12.77 | 1.15 | 883431798.0 | 18365.91 |
| 2023-11-16 | 1125.53 | 3.03 | 0.27 | 606149798.0 | 12724.74 |
| 2023-11-17 | 1101.19 | -24.34 | -2.16 | 1178535597.0 | 22779.45 |
| 2023-11-20 | 1103.66 | 2.47 | 0.22 | 751153197.0 | 15120.92 |

The data matched correctly with the figures displayed in the website.

<

- Second, the costs for our solution are monthly and yearly estimated in two scenarios as follows:

Scenario 1:

| Service | Region | Configuration summary | Monthly | First 12 months total | Currency |
|--|------------------|--|-----------|-----------------------|----------|
| Amazon Elastic Container Registry | Canada (Central) | DT Inbound: Internet (2 GB per month), DT Outbound: Internet (2 GB per month), Amount of data stored (2 GB per month) | 0.38 | 4.56 | USD |
| AWS Fargate | Canada (Central) | Operating system (Linux), CPU Architecture (x86), Average duration (30 minutes), Number of tasks or pods (5 per day), Amount of ephemeral storage allocated for Amazon ECS (20 GB), Amount of memory allocated (2 GB) | 4.13 | 49.56 | USD |
| AWS Lambda | Canada (Central) | Architecture (x86), Architecture (x86), Invoke Mode (Buffered), Amount of ephemeral storage allocated (512 MB), Number of requests (100 per month) | 0 | 0 | USD |
| Amazon EventBridge | Canada (Central) | Size of the payload (1 KB), Number of custom events (10 per month), Number of partner events (10 per month), Number of cross region events (10 per month), Number of bus-2-bus in the same account events (10 per month), Number of invocations (10 per month), Number of events (10 per month), Number of replayed events (10 per month), Number of requests (10 per month), Number of events (10 per month), Number of invocations (10 per month) | 0.0000067 | 0 | USD |
| S3 Standard | Canada (Central) | S3 Standard storage (5 GB per month), PUT, COPY, POST, LIST requests to S3 Standard (3000), GET, SELECT, and all other requests from S3 Standard (3000), Data returned by S3 Select (5 GB per month), Data scanned by S3 Select (5 GB per month) | 0.16 | 1.92 | USD |
| Data Transfer | Canada (Central) | DT Inbound: Internet (5 GB per month), DT Outbound: Internet (5 GB per month) | 0.45 | 5.4 | USD |
| AWS Glue ETL jobs and interactive sessions | Canada (Central) | Number of DPUs for Apache Spark job (10), Number of DPUs for Python Shell job (1) | 19.29 | 231.48 | USD |
| Amazon Aurora MySQL-Compatible | Canada (Central) | Aurora MySQL Cluster Configuration Options (Aurora Standard), Storage amount (10 GB) | 58.46 | 701.52 | USD |
| Standard topics | Canada (Central) | DT Inbound: Internet (1 GB per month), DT Outbound: Internet (1 GB per month), Requests (100 per month), HTTP/HTTPS Notifications (100 per month), EMAIL/EMAIL-JSON Notifications (100 per month), SQS Notifications (100 per month), Amazon Web Services Lambda (100 per month), Amazon Kinesis Data Firehose (100 per month), Publish and Delivery Message Scanning (1 GB per Month), Audit Reporting (1 GB per Month), The amount of outbound payload data scanned per month (1 GB) | 0.51 | 6.12 | USD |
| Amazon CloudWatch | Canada (Central) | Number of Metrics (includes detailed and custom metrics) (5), Standard Logs: Data Ingested (1 GB), Logs Delivered to CloudWatch Logs: Data Ingested (1 GB), Logs Delivered to S3: Data Ingested (1 GB), Number of Custom/Cross-account events (1000), Number of Lambda functions (10), Number of requests per function (30 per month) | 2.8859 | 34.63 | USD |

Refer to : <https://calculator.aws/#/estimate?id=ef9e35228281a08fe94b2c13d096e81c89b674ba>

Scenario 2:

| Service | Region | Configuration summary | Monthly | First 12 months total | Currency |
|--|------------------|---|---------|-----------------------|----------|
| Amazon Elastic Container Registry | Canada (Central) | DT Inbound: Internet (100 GB per month), DT Outbound: Internet (100 GB per month), Amount of data stored (100 GB per month) | 19 | 228 | USD |
| AWS Fargate | Canada (Central) | Operating system (Linux), CPU Architecture (x86), Average duration (30 minutes), Number of tasks or pods (20 per day), Amount of memory allocated (4 GB), Amount of ephemeral storage allocated for Amazon ECS (100 GB) | 22.44 | 269.28 | USD |
| AWS Lambda | Canada (Central) | Invoke Mode (Buffered), Architecture (x86), Architecture (x86), Number of requests (2000 per month), Amount of ephemeral storage allocated (1 GB) | 26.68 | 320.16 | USD |
| Amazon EventBridge | Canada (Central) | Size of the payload (1000 KB), Number of custom events (3000 per month), Number of partner events (3000 per month), Number of cross region events (10 per month), Number of bus-2-bus in the same account events (3000 per month), Number of invocations (3000 per month), Number of events (3000 per month), Number of replayed events (3000 per month), Number of requests (3000 per month), Number of events (3000 per month), Number of invocations (3000 per month) | 1.10216 | 13.23 | USD |
| S3 Standard | Canada (Central) | S3 Standard storage (10 TB per month), PUT, COPY, POST, LIST requests to S3 Standard (100000), GET, SELECT, and all other requests from S3 Standard (100000), Data returned by S3 Select (1 TB per month), Data scanned by S3 Select (1 GB per month) | 257.42 | 3089.04 | USD |
| Data Transfer | Canada (Central) | DT Inbound: Internet (10 TB per month), DT Outbound: Internet (1 TB per month) | 92.16 | 1105.92 | USD |
| AWS Glue ETL jobs and interactive sessions | Canada (Central) | Number of DPU for Apache Spark job (10), Number of DPUs for Python Shell job (1) | 1463 | 17556 | USD |
| Amazon Aurora MySQL-Compatible | Canada (Central) | Aurora MySQL Cluster Configuration Options (Aurora Standard), Storage amount (10 TB) | 1755.66 | 21067.92 | USD |
| Standard topics | Canada (Central) | DT Inbound: Internet (100 GB per month), DT Outbound: Internet (100 GB per month), Requests (1000 per month), HTTP/HTTPS Notifications (1000 per month), EMAIL/EMAIL-JSON Notifications (1000 per month), SQS Notifications (100 per month), Publish and Delivery Message Scanning (100 GB per Month), Audit Reporting (100 GB per Month), The amount of outbound payload data scanned per month (100 GB), Amazon Web Services Lambda (1000 per month), Amazon Kinesis Data Firehose (1000 per month) | 51 | 612 | USD |
| Amazon CloudWatch | Canada (Central) | Number of Metrics (includes detailed and custom metrics) (300), Standard Logs: Data Ingested (1 GB), Logs Delivered to CloudWatch Logs: Data Ingested (1 GB), Logs Delivered to S3: Data Ingested (1 GB), Number of Custom/Cross-account events (1000), Number of requests per function (30 per month), GetMetricData: Number of metrics requested (300), GetMetricWidgetImage: Number of metrics requested (300), Number of other API requests (300) | 91.3979 | 1096.77 | USD |

Refer to: <https://calculator.aws/#/estimate?id=adfc8e577322761bb7db94c9903d9585ecbbd184>

The scenario 1 was for data storage and processing data size under 5GB, and duration of Glue jobs (both Spark and Python shell) under 3 hours a month, which was expected to result in a monthly cost of 86.26 USD, and a yearly cost of 1,035.12 USD. In a scenario 2, the data size was expected to be 10 TB, and duration of Glue jobs was around 300 hours a month, which is likely to result in a monthly cost of 3,779.85 USD, and a yearly cost of 45,358.2 USD.

- Third, the development was conducted quickly.

AWS was able to provide different data engineering tools on the same platform to build an end-to-end data warehouse solution. This feature offers multiple benefits to development process. First, there is no requirement of installing software, licensing, and setting up environment from scratch. All of the setups were already done by AWS team. Regarding to the above experiment, developers do not need to buy physical servers, operating system license, or no need to install by themselves tools such as MySQL database, monitoring workflows tools (e.g. Airflow), file storage (e.g Hadoop), data batch processing tool (e.g. Spark), or messaging service (e.g. Kafka). As there are already alternative services in AWS such as Aurora, Cloudwatch, S3, Glue, and SNS. Second, there

will be less codes to be written. With the mentioned experiment, developers only need to know two programming languages such as Python, and SQL, which is used to develop Glue jobs and perform queries in data warehouse. While, for other services, developers can interact with a friendly UI to declare inputs in forms to create, and set up the tool. These two benefits help businesses to save time in development as developers just need to concentrate on writing codes for the core application, while other things such as infrastructure, and integration were supported by AWS.

6. Conclusion

In conclusion, AWS serverless solution can be a perfect alternative to on-premise solution in terms of developing data warehouses. AWS severless solution can provide multiple benefits, which can help it become a number one choice for businesses, such as the ease of integration and use. With less code and less infrastructure setup, a data warehouse project can have a high chance of becoming successful, reducing the risk of wasted investment. However, AWS serverless solution is not a perfect choice. As shown in the previous section, the costs of AWS solution can reach more than 40,000 USD a year, which can be more expensive than an on-premise solution (around 20,000 USD, including 5,000 USD for a physical server, 1,069\$ for Windows Server license, and 15,123 USD for Microsoft SQL Server Enterprise database license, while other infrastructure technologies can use open-source for free). Therefore, in case corporations are conscious of the costs, they should plan carefully and make comparisons between two solutions to choose to most appropriate option. On some occasions, corporations can think of utilizing multiple cloud services and on-premise solutions to figure out the best implementation for their applications.

7. References

Windows Server 2022 Licensing & Pricing | Microsoft. (n.d.). <https://www.microsoft.com/en-us/windows-server/pricing>

SQL Server 2022—Pricing | Microsoft. (n.d.). <https://www.microsoft.com/en-us/sql-server/sql-server-2022-pricing>

Stanovici, A. (2022b, July 11). *Calculating On-Premise vs Cloud Costs* — PMSquare. PMSquare. <https://pmsquare.com/analytics-blog/2022/6/9/calculating-on-prem-vs-cloud-costs>

What is IAM? - AWS Identity and Access Management. (n.d.). <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>

What is Amazon Elastic Container Registry? - Amazon ECR. (n.d.).

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html?pg=ln&sec=hs>

What is Amazon Elastic Container Service? - Amazon Elastic Container Service. (n.d.).

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>

What is AWS Lambda? - AWS Lambda. (n.d.-b).

<https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>

What is Amazon CloudWatch? - Amazon CloudWatch. (n.d.).

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html>

What is Amazon EventBridge? - Amazon EventBridge. (n.d.).

<https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html>

What is Amazon SNS? - Amazon Simple Notification Service. (n.d.).

<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>

What is Amazon S3? - Amazon Simple Storage Service. (n.d.).

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>

What is AWS Glue? - AWS Glue. (n.d.). <https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html>

What is Amazon Aurora? - Amazon Aurora. (n.d.).

https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_AuroraOverview.html

what-is-aws. (n.d.). [Video]. Amazon Web Services, Inc. <https://aws.amazon.com/what-is-aws/>

8. Appendix

8.1. Lambda Function - **MyDemoFunction**:

```
import boto3
import time
import pandas as pd
import requests
import json
import awswrangler as wr

def lambda_handler(event, context):

    client = boto3.client('ecs')

    cluster_name = 'web_scraping_api_demo'
    task_family = 'run_web_scraping_api_demo'
    subnet1 = 'subnet-0d4eda1f233c858c2'
    subnet2 = 'subnet-0ea9853bd0f5d3e88'
    subnet3 = 'subnet-0b1c8cd53b0da8492'
    securityGroup = 'sg-0374225503e692273'
```

```

s3_bucket = 'vnstockmarket-sample-dev'
sub_path = 'stg_vnindex_historical_index/'

response = client.run_task(
    cluster=cluster_name,
    taskDefinition=task_family,
    launchType='FARGATE',
    networkConfiguration={
        'awsvpcConfiguration': {
            'subnets': [
                subnet1,
                subnet2,
                subnet3,
            ],
            'securityGroups': [
                securityGroup,
            ],
            'assignPublicIp': 'ENABLED'
        }
    }
)

task_arn = response['tasks'][0]['containers'][0]['taskArn']
print(task_arn)
time.sleep(60)
response = client.describe_tasks(
    cluster=cluster_name,
    tasks=[
        task_arn,
    ],
)
private_ip =
response['tasks'][0]['containers'][0]['networkInterfaces'][0]['privateIpv4Add
ress']
eni_id = None
for detail in response['tasks'][0]['attachments'][0]['details']:
    if 'eni-' in detail['value']:
        eni_id = detail['value']

eni = boto3.resource('ec2').NetworkInterface(en_i_id)
public_ip = eni.association_attribute['PublicIp']
print(private_ip, eni_id, public_ip)

BASE = f"http://{public_ip}:5000/"

response = requests.get(BASE + 'data')
response = json.loads(response.json())
df = pd.DataFrame(response)
print(df)
filename = df['RECORDED_DATE'].values[0]
filename = filename.split("/")
filename = [filename[len(filename) - i - 1] for i in
range(len(filename))]
filename = ''.join(filename)
destination = f"s3://{s3_bucket}/{sub_path}{filename}.csv"
print(destination)
wr.s3.to_csv(df, destination, index=False)

```

```

# Stop ECS task
client.stop_task(
    cluster=cluster_name,
    task=task_arn,
)

return {
    "statusCode": 200,
    "body": json.dumps({
        "message": "The today VNINDEX has been stored in S3!",
    }),
}

```

8.2. Glue job - clean:

```

from datetime import datetime
import pytz
import numpy as np
from dateutil.relativedelta import relativedelta
from pyspark.sql.functions import *
from pyspark.sql.functions import col, lit
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext, DynamicFrame
from awsglue.job import Job

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)

today = datetime.now(pytz.timezone("Asia/Bangkok")).date() #
"America/Toronto"
extract_date = today # able to edit datetime.strptime("yyyymmdd", "%Y%m%d")
bucket = "vnstockmarket-sample-dev"
table_name = "stg_vnindex_historical_index"

delta = relativedelta(days=1)
for date in np.arange(extract_date, today + delta, delta):
    date = date.astype(datetime)
    date_string = date.strftime("%Y%m%d")
    print(f"Processed Date: {date_string}")

    # source S3 path
    s3_stg_vnindex_historical_index_path =
f"s3://{bucket}/{table_name}/{date_string}.csv"

    # destination S3 path
    s3_cleaned_stg_vnindex_historical_index_path =
f"s3://{bucket}/cleaned/{table_name}/{date_string}.csv"

    stg_vnindex_historical_index =
spark.read.format("csv").option("header", "true")

```



```

).load(s3_stg_vnindex_historical_index_path)
stg_vnindex_historical_index = stg_vnindex_historical_index.select(
    to_date(col('RECORDED_DATE'),
'dd/MM/yyyy').alias('RECORDED_DATE'),
    col('CLOSE_PRICE').cast('double').alias('INDEX_SCORE'),
    regexp_replace(col('INDEX_CHANGE').substr(lit(1),
instr(col('INDEX_CHANGE'), '(') - 1),
    ',',''),
    ').cast('double').alias('INDEX_CHANGE_VALUE'),
    regexp_replace(
        regexp_replace(
            col('INDEX_CHANGE').substr(instr(col('INDEX_CHANGE'),
'(') + 1,
length(col('INDEX_CHANGE')) -
instr(col('INDEX_CHANGE'), '('))
, ' %\\', '')
, 'INFINITY',
    ').cast('double').alias('INDEX_CHANGE_PERCENT'),
    col('MACHED_VOLUME').cast('double').alias('MACHED_VOLUME'),
    col('MACHED_AMOUNT').cast('double').alias('MACHED_AMOUNT'),

col('SELF_DEALED_VOLUME').cast('double').alias('SELF_DEALED_VOLUME'),

col('SELF_DEALED_AMOUNT').cast('double').alias('SELF_DEALED_AMOUNT'),
    col('OPEN_PRICE').cast('double').alias('OPEN_PRICE'),
    col('HIGHEST_PRICE').cast('double').alias('HIGHEST_PRICE'),
    col('LOWEST_PRICE').cast('double').alias('LOWEST_PRICE')
)
stg_vnindex_historical_index.show()

stg_vnindex_historical_index.toPandas().to_csv(s3_cleaned_stg_vnindex_histori
cal_index_path, index=False)
print(f"The {table name} data of {date string} has been cleaned!")

```

8.3. Glue job - increment:

```

from datetime import datetime
import pytz
import numpy as np
from dateutil.relativedelta import relativedelta
from pyspark.sql.functions import *
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext, DynamicFrame
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrame
import mysql.connector

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)

```

[illegible]

```
) .load(s3_cleaned_stg_vnindex_historical_index_path)
    print(f"Cleaned data of {date_string} has been loaded from S3!")

    # append new/rerun data records
    tmp_stg_vnindex_historical_index =
tmp_stg_vnindex_historical_index.union(
    cleaned_stg_vnindex_historical_index)

# close connection to Aurora
cnx.close()

tmp_stg_vnindex_historical_index.show(1)
num_appended_rows = tmp_stg_vnindex_historical_index.count()
print(f"({num_appended_rows}) rows will be appended!")

# overwrite Aurora table
tmp_stg_vnindex_historical_index.write.format('jdbc').options(
url=f"jdbc:mysql://{hostname}:{port}/{database_name}",
    driver="com.mysql.jdbc.Driver",
    dbtable=f"{aurora_table}",
    user=username,
    password=pwd).mode('append').save()
```